НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В.Коваль
(підпис) (ініціали, прізвище)
“___”_____2019 р.

ДИПЛОМНА РОБОТА
на здобуття ступеня бакалавра

з напряму підготовки
6.050103 “Програмна інженерія’’

на тему: Система поліпшення проведення занять (клієнт для iOS)

Виконав: студент 4 курсу, групи ТІ-51

Мельниченко Кирило Сергійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент, к.т.н. Ходаківський Олександр Володимирович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 6.050103 “Програмна інженерія’’

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019 р.

ЗАВДАННЯ
на дипломну роботу студенту
Мельниченку Кирилу Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи “ Система поліпшення проведення занять (клієнт для iOS)”

керівник роботи доцент, к.т.н. Ходаківський Олександр Володимирович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”___”_____ 201__ р. № _____

2. Строк подання студентом роботи ___ _____ 201__ р.
3. Вихідні дані до роботи персональний комп’ютер під керуванням операційної системи macOS, мова програмування Swift, інтегроване середовище розробки Xcode
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати існуючі програмні рішення та можливі засоби реалізації додатку, обґрунтувати обрані програмну архітектуру системи та шляхи розробки програмних додатків, розробити програмне забезпечення, розробити користувацький інтерфейс, зробити висновки за результатами роботи
5. Перелік ілюстраційного матеріалу (з точним зазначенням обов’язкових креслень)
1. Архітектура VIPER 2. Архітектура MVC. 3.Підключення фреймворку CoreData для використання локальної бази даниї 4. Робота з локальною базою даних. 5.Шаблон проектування – делегування. 6. Інтерфейс додатку.

Дата видачі завдання ”___”_____ 201__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі		
2.	Розробка архітектури та загальної структури системи		
3.	Розробка структур окремих модулів		
4.	Підготовка матеріалів		
5.	Програмна реалізація системи		
6.	Захист програмного продукту		
7.	Оформлення пояснювальної записки		
8.	Передзахист		
9.	Захист		

Студент _____ Мельниченко К.С
(підпис) (прізвище та ініціали)

Керівник роботи _____ Ходаківський О. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Мета роботи — розробка програмного продукту для платформи iOS, який сприятиме покращенню проведення занять. А саме зробити акцент на взаємодії слухачів та лекторів під час лекції та після її завершення. Для розробки інтерфейсів було обрано storyboard та Interface Builder, який вбудовано в інтегроване середовище розробки Xcode. Для розробки додатка використовувалась мова програмування Swift та фреймворки CoreData, FirebaseDatabase та FirebaseAuth. Також було обрано архітектури VIPER та MVC для побудови модулів системи.

Записка містить 51 сторінока, 20 рисунків, 2 таблиці, 3 додатка і 16 посилань.

Ключові слова: IOS, APPLE, SWIFT, XCODE, ЛОКАЛЬНА БАЗА ДАНИХ, COREDATA, FIREBASE, АРХІТЕКТУРА VIPER, АРХІТЕКТУРА MVC.

ABSTRACT

The purpose of the work is to develop a software product for the iOS platform, which will help to improve the training. Namely, to focus on the interaction of listeners and leaders during and after the lectures. For designing interfaces was chosen the storyboard and Interface Builder, which is integrated into an integrated development environment for Xcode. Swift programming language and CoreData, FirebaseDatabase and FirebaseAuth frameworks were used to develop the application. The VIPER and MVC architectures were also selected to build system modules.

The note contains 51 pages, 20 figures, 2 tables, 3 attachments and 16 links. KEYWORDS: IOS, APPLE, SWIFT, XCODE, LOCAL DATABASE, COREDATA, FIREBASE, ARCHITECTURE VIPER, ARCHITECTURE MVC.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ARC — Automatic Reference Counter.

IDE — Integrated Development Environment.

DB — Database.

CLR — Common Language Runtime.

IL — Intermediate Language.

GCD — Grand Central Dispatch.

API — Application Programming Interface.

GUI — Graphical User Interface.

MVC — Model-View-Controller.

VIPER — View-Interactor-Presenter-Entity-Router.

UIKit — бібліотека для роботи с користувацьким інтерфейсом в iOS.

DD — Dynamic Dispatch.

OQ — Operation Queue.

CD — Core Data.

CP — CocoaPods.

SOLID — принципи побудови ахрітектури.

ВСТУП

За останні роки роль сучасних технологій в житті люди дуже змінилась. Важко уявити зараз будь-який процес виробництва, або ж ведення документів без використання спеціальних програм. Навіть у повсякденному житті люди майже не розлучні зі своїми гаджетами. Але чомусь ще й досі люди практично не використовують їх в сфері навчання, а саме в проведенні занять.

На сьогоднішній день, в нашій країни та взагалі у всьому сучасному світі – не вистачає комп'ютерних систем, які б могли допомогти автоматизувати (частково або повістю) та поліпшити проведення занять в університетах, школах та навіть на приватних курсах.

Мобільні телефони вже давно стали невід'ємною частиною нашого життя. Наш телефон вже більше не є одним з пересічних пристроїв у нашому житті. Телефони стали нашими електронними біл-бордами, демонструючи хто і що для нас важливе. Після цього важко уявити, чому вони ще й досі не використовуються під час проведення занять та й взагалі усього учбового процесу.

Тому, було запропоновано дослідити, як саме можна організувати впровадження програмного продукту в процес проведення занять та реалізувати її. При чому реалізувати даний продукт як додаток для мобільного телефону, який використовує операційну систему iOS.

На даному етапі роботи було зроблено акцент на взаємодії студентів з викладачем під час лекцій. В майбутньому ставитиметься мета охопити повну взаємодію, тобто — виставлення балів, відмітки про присутність. Так як, зараз частіше за все лекції відбуваються у великих аудиторіях, де знаходиться безліч людей і тому у слухачів (наприклад – студентів) просто не має можливості поставити питання лектору.

Розроблений додаток є частиною з кросплатформної програмної системи, тобто він буде доступний не тільки на платформі iOS, а і у web версії, та на платформі Android.

Даний програмний застосунок розроблено за допомогою мови програмування, розробленою компанією “Apple” Swift. Ця мова програмування є однією з найшвидших сучасних мов. Для розробки було обрано IDE – Xcode десятої версії. Також в додатку використовувалися такі засоби, як CoreData, FirebaseDatabase, FirebaseAuth.

В першому розділі чітко сформульовано завдання цієї роботи. Другий розділ містить в собі дослідження вже існуючих програмних застосунків за схожою тематикою. В третьому розділі розглянуто основні засоби розробки додатка. Четвертий розділ містить ключові аспекти проектування архітектури та програмної реалізації системи. В п’ятому розділі описується взаємодія користувача з розробленим додатком для платформи iOS.

1. РОЗРОБКА ДОДАТКА ДЛЯ ОПЕРАЦІЙНОЇ СИСТЕМИ IOS СПРЯМОВАНОГО НА ПОЛІПШЕННЯ ПРОВЕДЕННЯ ЗАНЯТЬ.

Зараз дуже важливо, щоб система, яка розроблювалась, була доступна весь час. Важко уявити людину без мобільного телефону, тому саме і обрано розробку даної системи для платформи iOS.

Вважається, що дана операційна система є кращою серед мобільних операційних систем, так як вона є закритою, що гарантує надійність і безпеку для особистих даних користувача, а також в неї інвестується багато коштів, щоб продовжувати тримати заданий гідний рівень якості.

1.1. Призначення та користувачі

Дане програмне забезпечення призначене для поліпшення і часткової автоматизації проведення занять в університетах та на курсах і навіть для дистанційного проведення лекцій. За допомогою даного додатку для платформи iOS, власники мобільних телефонів від компанії “Apple” матимуть змогу вільно поставити питання, які їх цікавлять .

Досить важлио, що при постановці питання, слухачі не будуть відволікати лекторів від проведення заняття. Викладачі ж в свою чергу зможуть відповісти на всі питання після лекції, або ж якщо на питання була уже дана відповідь іншим слухачем, то просто відмітити відповідь, як вірну.

Кінцевими користувачами можуть бути викладачі та студенти вищих навчальних закладів або курсів. Також можливе використання даного застосунка при організації дистанційного навчання. так як програмна система не потребує зосередженню всіх дійових осіб в одній аудиторії.

1.2. Задачі, які розв’язуються програмним забезпеченням

Було поставлено задачу порівняти вже існуючі системи, виділити головні їх переваги та недоліки та за допомогою цієї інформації спроектувати систему, яка б могла поліпшити процес проведення занять.

Головною задачею даного програмного забезпечення є допомогти поліпшити та частково автоматизувати процес проведення занять. Для цього потрібно зробити акцент на взаємодії студентів з викладачем.

Тому спочатку було зроблено акцент саме на проведенні лекційних занять. Зараз лекції проводяться у великих аудиторіях, де знаходиться безліч людей. Саме через це у слухача іноді просто не має можливості запитати лектора. Даний додаток буде розв’язувати саме цю проблему комунікації викладача з студентами під час занять.

1.3 Задача розробки додатка для платформи iOS

В попередніх розділів було розглянуто мету цієї роботи. Спираючись на факти того, що зараз просто не вистачає програмних систем, що можуть сприяти покращенню проведенню занять, значної ролі новітніх технологій в житті людини, та проблем комунікації між слухачами та лекторами під час лекцій, було сформовано завдяння для виконання роботи.

Завдання :

1. Розробити додоток для платформи iOS, який спрямований на покращення проведення занять.
2. Використати для розробки мову програмування Swift та інтегровану середу розробки Xcode.
3. Забезпечити анонімність користувачів.
4. Дати змогу вільно ставити питання під час лекційних занять та по їх завершенню.
5. Передбачити можливість відповідати на поставлені питання.

6. Забезпечити можливість відмічати відповідь, як вірну.

Висновки до розділу 1

В даному розділі було розглянуто призначення додатка, потенційні користувачі, задачі, які розв’язуються програмною системою. Також було чітко сформульовано завдання роботи.

2. ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ З ВПРОВАДЖЕННЯ СУЧАСНИХ ТЕХНОЛОГІЙ В ПРОВЕДЕННЯ ЗАНЯТЬ

Дослідження вже існуючих програмних систем є дуже важливим етапом в плануванні та формулюванні точного технічного завдання, збору вимог до програмного продукту та проектуванні системи.

Не провівши попередній аналши схожих застосунків, можна зробити ті ж самі помилки, що і програмісти, які розроблювали ці застосунки. Також це є гарним способом для виявлення корисних функцій або інтерфейсних рішень.

2.1. Існуючі програмні засоби

На даний момент аналогічних систем або додатків майже не існує. Після детальних пошуків вдалося знайти лише один додаток для iOS - iSchool, який нагадує електронний щоденник. Але головною відмінністю є те, що всі данні в ній зберігаються локально на мобільному девайсі і заповнюються конкретно самим користувачем, відсутня будь-яка синхронізація з сервером та іншими користувачами (студентами або викладачами) .

Очевидно, що цей програмний застосунок розроблювався, для того, щоб школярі могли використовувати свій мобільний телефон зі встановленим додатком, замість свого паперового щоденника. Такий підхід не є зовсім вдалим, тому що він обумовлює використання даної програмноє системи тільки школярами.

Також було знайдено web-сайт, який використовується тільки одним проектом — платними курсами від компанії “Badoo” для забезпечення належного проведення дистанційного навчання. Головним недоліком цієї системи вважається лише те, що вона доступна тільки у web версії, що позбавляє користувачів можливості використання цієї програмної системи без наявного підключення до мережі інтернет з належною швидкістю.

Дана програмна система також орієнтована тільки для користувачів лише конкретних курсів. До того ж вона не задовільнить користувачів, які надають переваги встановленню додатків на свій мобільний телефон, а не використанню web версій.

2.1.1 Недоліки та головні відмінності існуючих програмних застосунків

Основною відмінністю розглянутих програмних застосунків є те, що вони розроблялися не з цілю покращити проведення занять та частково їх автоматизувати, а для того, щоб виконувати якусь одну функцію (ведення щоденника, або нотатки проходження курсів). Також дані додатки та web-системи не планують виходити на глобальний ринок та впроваджувати свої ідеї взагалі в сучасну освіту.

Недоліки вже існуючих систем:

— Орієнтація на конкретний заклад або курси.

— Частіше за все, доступно для використання тільки web версія, без наявних додатків для мобільних телефонів та програм на персональний комп’ютер або ноутбук.

— Обмежений функціонал.

— Відстуність будь-якої комунікації між користувачами.

2.2 Результати досліждень

Після проведення детального аналізу схожих систем, було встановлено, що потрібно зробити додаток в якому:

— Наявна авторизація та реєстрація для користувачів.

— Наділено користувачів за групи “Викладачі” додатковими можливостями.

— Передбачена можливість використання додатку без підключення до інтернету.

— Можливий перегляд попереднього матеріалу.

— Розроблений дружній інтерфейс

— Адаптивний дизайн для iPhone та iPad.

— Передбаченно можливість перегляду власного профіля.

Висновки до розділу 2

В даному розділі було розглянуто вже існуючі програмні рішення в сфері впровадження нових технологій для покращення проведення занять. Було виявлено основні їх недоліки та на основі цих даних встановлено, що саме необхідно реалізувати в розроблюваній системі.

3. ЗАСОБИ РОЗРОБКИ

Розробка програм на операційні системи від компанії “Apple” є доволі витратною, адже потребує комп’ютера, на якому встановлена macOS, сертифіката від Apple Developer Center. Але компанія в свою чергу гарантує якісну середу розробки з використанням всіх нових можливостей iOS, macOS, tvOS, watchOS, повним набором емуляторів iPhone та iPad, та наявною у вільному доступі документацією щодо створення додатків за допомогою мов Swift та Objective-C.

Для розробки даного програмного продукту використовувався ноутбук від компанії “Apple” – MacBook Pro, з встановленою останньою версією операційної системи macOS Mojave. Даний комп’ютер було обрано, тому що розробка додатків для платформи iOS вимагає саме цю версію операційної системи. В якості мови було вибрано мову програмування Swift. Роботу з локальною базою даних було організовано за допомогою фреймворку CoreData, в якості серверної бази даних було обрано Firebase.

3.1. Середовище розробки Xcode

Компанія “Apple” пропонує лише одне середовище розробки для створення додатків для операційної системи iOS, яке має назву Xcode. Xcode — інтегроване середовище розробки (IDE) виробництва Apple. Перша версія випущена в 2001 році, а подальші стабільні версії поширюються безкоштовно через Mac App Store. Xcode включає в себе більшу частину документації розробника від Apple і Interface Builder - додаток, що використовується для створення графічних інтерфейсів. Пакет Xcode містить змінену версію вільного набору компіляторів GNU Compiler Collection і підтримує мови C, C++, Objective-C, Swift, Java, AppleScript, Python і Ruby з різними моделями програмування, включаючи (але не обмежуючись) Cocoa, Carbon і Java.

Для рішення конкретно цієї задачі було обрано середовище розробки Xcode останньої (10.2.1) версії, тому що тільки ця версія підтримує розробку програм та додатків за допомогою мови програмування Swift 5.0.

3.2. Мова програмування Swift

Swift — багатопарадигмова компільована мова програмування, розроблена компанією Apple для того, щоб співіснувати з Objective C і бути стійкішою до помилкового коду. Swift була представлена на конференції розробників WWDC 2014.

Розробку мови Swift почав Chris Lattner у співпраці із багатьма іншими програмістами. Ідеї для Swift запозичені із таких мов програмування як Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, та багатьох інших зі списку. Ця мова вже має за плечима роки розвитку, і вона продовжує розвиватися, включаючи в себе все нові і нові можливості[5].

Компілятор Swift побудований з використанням технологій вільного проекту LLVM. Swift успадковує найкращі елементи мов C і Objective-C, тому синтаксис звичний для знайомих з ними розробників, але водночас відрізняється використанням засобів автоматичного розподілу пам'яті і контролю переповнення змінних і масивів, що значно збільшує надійність і безпеку коду.

Мова також пропонує безліч сучасних методів програмування, таких як замикання, узагальнене програмування, лямбда-вирази, кортежі і словникові типи, швидкі операції над колекціями, елементи функційного програмування. Основним застосуванням Swift є розробка користувацьких застосунків для macOS, iOS, tvOS, watchOS з використанням інструментів Cocoa і Cocoa Touch. При цьому Swift надає об'єктну модель, сумісну з Objective-C. Сирцевий код мовою Swift може змішуватися з кодом на C і Objective-C в одному проекті.

Swift – це перша потужна мова програмування, така ж зрозуміла і захоплююча, як скриптова мова. Вона підтримує так звані playground-и, які дозволяють програмістам експериментувати з кодом, бачачи результат в режимі реального часу без необхідності компілювати і запускати додаток[9].

Swift виключає великк кількість поширених програмних помилок за допомогою застосування сучасних програмних паттернів.

3.3. Конструктор інтерфейсів в Xcode

Interface Builder - додаток від Apple для операційної системи Mac OS X. Він є частиною Xcode (колишній Project Builder), спеціальної системи інструментів для розробників Apple Developer Connection[7].

Interface Builder вперше з'явився в 1986 і був реалізований за допомогою мови Lisp. Interface Builder був задуманий і розроблений Джин-Мері Холлотом з використанням інструментів об'єктно-орієнтованого програмування в ExperLisp і глибоко інтегрований з інструментами Macintosh[10].

Interface Builder надає палітри, або колекцій, об'єктів користувальницького інтерфейсу для Objective-C і Swift розробників. Ці об'єкти користувацького інтерфейсу містять такі елементи, як текстові поля, таблиці даних, слайдери і спливаючі меню. Палітри Interface Builder є повністю розширюваними, тобто будь-який розробник може розробляти нові об'єкти і додавати їх до палітри IB.

Для створення інтерфейсу, розробник просто перетягує елементи інтерфейсу з палітри на вікно або меню. Конкретні об'єкти, які отримують повідомлення вказуються в коді програми. Таким чином все ініціалізації відбуваються до виконання, що веде до підвищення продуктивності.

3.4. Фреймворк CoreData

Core Data – фреймворк від компанії Apple, вбудований в операційну систему iOS, MacOS, який дозволяє розробнику взаємодіяти з базою даних[11]. Був представлений компанією Apple з анонсом Mac OS X 10.4 Tiger і iOS з iPhone SDK 3.0[12].

Даний інструмент дозволяє даними бути організованими в Сутність-Атрибут-Значення. Управляти цими даними доволі просто, за допомогою маніпуляцій сутностями, як об’єктами звичайних класів, створених безпосередньо в коді.

Core Data описує дані, які зберігаються в iOS додатку, код може використовувати для збереження і записи даних в додатку. Модель БД створюється в

Interface Builder. Код пишеться на Objective-C або Swift. Core Data організований в величезні класи.

Структура :

— Managed Object Context - компонент з яким відбувається взаємодія, кожен раз, коли йде збереження або оновлення.

— Persistent Store Coordinator - виконує зберігання даних. Посередник між базою даних і контекстом

— Persistent Store - сховище, де зберігаються дані попередньо записані за допомогою Core Data.

— Managed Object Model - модель бази даних.

Core Data може конвертувати дані в XML, бінарний код, SQLite для зберігання в гаджеті на операційних системах iOS або macOS.Навіть, якщо на комп'ютері відсутній Xcode – завжди існує переглянути файли бази. Звичайно, цей факт розповсюджується лише на власників комп'ютерів від компанії "Apple".

Основна частина фреймворка була написана під час роботи Стіва Джобса в компанії NeXT, Enterprise Objects Framework (EOF) на мові Objective-C[15].

Але існує один недолік при роботі з даним інструментом – це швидкість виконнаня методів об'єктами з бази даних. Даний факт пояснюється тим, що класи з Core Data використовують Objective-C Runtime.

Також замість статичної диспетчеризації використовується динамічна, що теж робить свій внесок в швидкість роботи та виконання методів об'єктами цих класів.

3.4. Менеджер залежностей CocoaPods

CocoaPods — менеджер залежностей для проектів, написаних на мові Swift або Objective-C[10].

Даний менеджер містить в собі тисячі бібліотек і допомагає розробникам легко та елегантно масштабувати проекти. В цілому його ціль — покращити доступність та участь в сторонніх бібліотеках, код яких знаходиться у відкритому доступі. Також

однією з задач CocoaPods є створення єдиної централізованої системи залежностей у проекті.

Залежності для проектів вказуються в одному текстовому файлі, який називається “Podfile”. Доступний також файл залежностей з розширенням .lock, даний файл оновлюється кожен раз після внесення змін в основний Podfile.

Цей менеджер є дещо схожим на Maven для Java, або Ruby Gems для Ruby. Ідеї яких близькі до ідей CocoaPods, а саме — після того, як додається нова бібліотека до проекту, не потрібно слідкувати за її оновленнями, так як менеджер залежностей буде виконувати все це автоматично без необхідного втручання розробника.

Досить важливим є також той факт, що CocoaPods має прямий зв’язок до основного файлу проекту .xcodeproj. Це обумовлено тим, що використання сторонніх бібліотек в середовищі розробки Xcode вимагає формування нового файлу запуска проекта — . xcworkspace. Даний менеджер вирішує також і цю проблему.

3.5. UIKit

UIKit — це один з фундаментадьних інструментів розробки додатків для платформи iOS. Він використовується для побудови інтерфейсів застосунка.

Класи які знаходятся в цій бібліотеці[14]:

- допоміжні елементи – UIEvent, UIFont, UIColor, UIDevice, UIScreen, UIImage;
- макети, які можуть прокручуватись UIScrollView , UITableView, UICollectionView[13];
- контроли – UITextField, UIButton, UIDatePicker, UIPageControl, UISlider, UISlider[14];
- візуальні елементи – UIWindow, UILabel, UIPickerView, UIImageView, UIWebView, UIToolbar;
- контролери видів – UIViewController, UINavigationController, UITabBarController;
- відповідачі – UIResponder та UIApplication.

За допомогою цих класів і будується інтерфейс. Наприклад, клас UIViewController використовується в будь-якому модулі в якому є відображення даних на екрані. Компанія “Apple” радить використовувати саме контролери видів з цієї бібліотеки, тому що життєвий цикл контролера тісно пов’язаний з життєвим циклом вида.

Основою будь-якого додатка для платформи iOS становлять таблиці. Даний факт обумовлюється тим, що це зручний елемент для відображення великої кількості схожих даних. В цій бібліотеці передбачена можливість відображення таблиць двох видів – згруповану таблицю та плоску. Також кожна таблиця може мати свій заголовок зверху таблиці та колонтитул знизу. Ці два елемента мають властивість закріплення на початковій позиції, тобто під час прокручування вони будуть залишатися на одному місці.

В деяких випадках потрібно використовувати UICollectionView, якщо потрібно організувати горизонтальне прокручування або ж потрібно розмістити елементи більш ніж в один стовбець.

В бібліотеці UIKit вже передбачено чотири типи комірок для таблиць, тобто розробникам програмного забезпечення не потрібно витрачати свій час на побудову деяких базових речей за допомогою цієї бібліотеки.

Комірки доступні в UIKit по замовчуванням:

1. Basic (Звичайна комірка з одним полем для тексту).
2. Subtitle (Комірка з двома текстовими полями, один з яких менший)
3. Right Detail (Комірка з іконкою в правій стороні)
4. Left Detail (Комірка з іконкою зліва)

Також передбачена можливість додати навігаційне або пояснювальне зображення для будь-якого з цих типів комірок.

За допомогою об’єктів класу типу UIResponder відбуваєть відловлення жестів та дій користувача. Спочатку методи цього класу викликаються на першому об’єкті (виді) по ієрархії та йому передається відповідна дія або жест. Якщо даний об’єкт не може опрацювати дію або жест, наприклад тому що, об’єкт не доступний, або

невидимий (атрибут даного об’єкта – IsHidden = true) , то тоді дія або жест передається наступному елемнету по ієрархії.

Висновки до розділу 3

В даному розділі описано засоби, використані при розробці додатку. Так як розроблене програмне забезбепечення призначене для роботи на платформі iOS, тому були вибрані саме такі засоби розробки.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

В наступних розділах буде розглянуто основні аспекти реалізації даного програмного забезпечення.

4.1. Архітектура додатку

Проектування системи є головним етапом в розробці додатків з того часу, як програмування стало об’єктно орієнтованим. Ця тенденція обумовлена важливістю наявності засобів для швидкого розширення програмного продукту без пошкодження чи зміни вже існуючого функціонала.

Для розробки використовувалися два архітектурних шаблони – VIPER та MVC. Про це детальніше йдеться в розділах 4.1.1 та 4.1.2.

В результаті розробки, проект додатку має наступний вигляд (Рисунок 4.1):

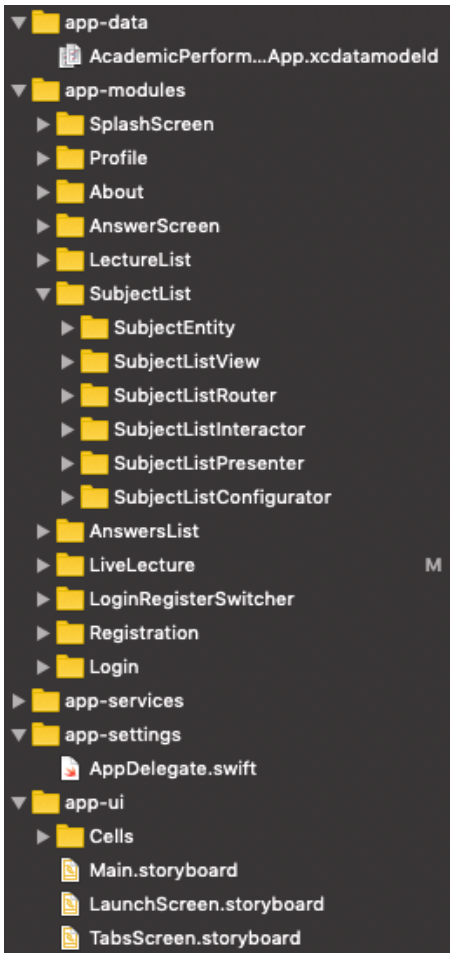

Рисунок 4.1 – Дерево проекта

Проект складається з трьох файлів, розширенням .storyboard, в яких знаходяться екрани з користувацьким інтерфейсом та логікою переходів між ними, файлу моделі даних для збереження в локальну базу даних, декількох модулів. Прийнято вважати один екран – це один модуль.

Класи в модулі згруповані та взаємодіють між собою за шаблонами проектування VIPER та MVC. Шаблон VIPER було обрано для модулів, які вимагають містити в собі багато різного функціонала, наприклад – завантажування даних і запис їх до локальної бази даних. Шаблон MVC обирався для модулів, які мали відповідати лише за відображення даних.

4.1.1. Архітектурний шаблон VIPER

У пошуку кращого способу спроектувати iOS додаток в стилі Clean Architecture, як описав Роберт Мартін, було запропоновано компанією “Rambler & Co” використовувати архітектуру VIPER, що на даний момент часу вважається кращим рішенням цієї проблеми. Чиста архітектура ділить логічну структуру програми на різні рівні обов'язків. Це спрощує ізолювання залежності (наприклад, база даних) і тестування взаємодії на границях між рівнями, підвищує рівень абстракції, знижує зв'язаність коду і вирішує проблеми маршрутизації в додатку[6].

Зв’язки в даній архітектурі :

— View – вид – відповідає за відображення даних на екрані і оповіщає Presenter про дії користувача. Пасивний, сам ніколи не запитує дані, тільки отримує їх від представника[4].

— Interactor – взаємодіючий – містить всю бізнес-логіку, необхідну для роботи поточного модуля.

— Presenter – представник – отримує від View інформацію про дії користувача і перетворює її в запити до Router'a, Interactor'a, а також отримує дані від Interactor'a, готує їх і відправляє View для відображення[4].

— Entity – сутність –об'єкти моделі, що не містять ніякої бізнес-логіки.

— Router – маршрутизатор – відповідає за навігацію модуля.

Зв'язки між прошарками в архітектурі VIPER (Рисунок 4.2) :

View – вид – має в собі посилання на протокол (інтерфейс) представника. Представник, в свою чергу містить посалання на протоколи вида, взаємодіючога і маршрутизатора. Маршрутизатор має в собі посилання на вид для забезпечення переходів міє модулями .Interactor – взаємодіючий – знає про існування представника за допомогою протокола. Також він містить в собі необхідні сервіси для реалізації бізнес-логіки додатку.

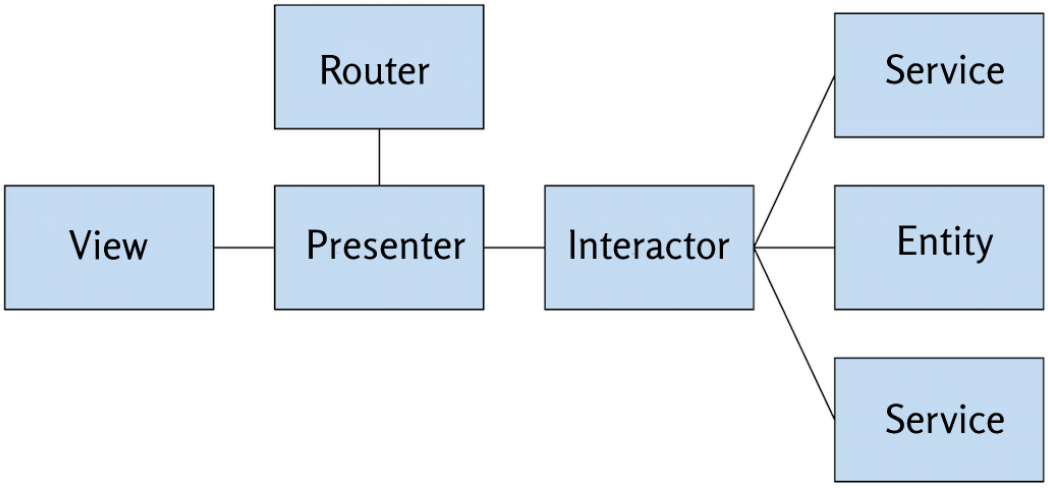

Рисунок 4.2 – Архітектура VIPER модуля

Перевагою такого модуля також можна назвати те, що він може бути покритим тестами на 90-100 %.

Головні характеристики архітектури VIPER :

— Допомагає слідувати SOLID принципам при розробці[16].
— Ділить модуль на більшу кількість окремих прошарків, що забезпечує принцип єдиної відповідальності.
— Полегшення написання тестів.
— Надає можливості покривати тестами модуль на 90-100%.
— Закриває кожен прошарок протоколом (інтерфейсом), тобто напряму прошарки не можуть взаємодіяти між собою.

— Зменшує швидкість розробки через опис додаткових сутностей модулів системи.

4.1.2. Архітектурний шаблон MVC

Модель–вигляд–контролер (або Модель–представлення–контролер. Model-view-controller, MVC) — архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення[3].

Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача[3].

В даному випадку використовувався стандартний шаблон MVC від компанії “Apple” з пасивною моделлю для відображення даних. Розглянемо цей шаблон в модулі додавання нової відповіді на запитання (Рисунок 4.3):

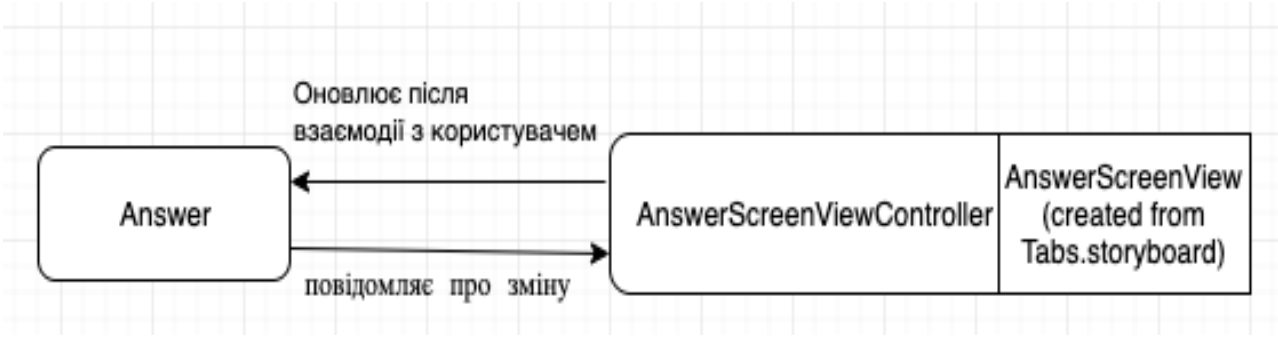

Рисунок 4.3 – Архітектура MVC для модуля додавання нової відповіді

Вид для даного модуля створений за допомогою Interface Builder і знаходиться він в файлі Tabs.storyboard. До виду приєднується контролер також за допомогою конструктора інтерфейсу. В контролері знаходиться модель, яку він змінює після взаємодії користувача з видом.

Голові характеристики архітектури MVC від Apple:

— С початку розробки — збільшення її швидкості.

— Важко тестувати великі модулі.

— Легкість для розуміння.

— Переважно використовується з пасивною модулью.

Саме через ці характеристики, було вирішено використовувати дану архітектуру тільки в модулях які відповідають тільки за відображення даних і не виконують ніяку складну роботу.

4.1.3. Шаблон проектування Делегування

В наш час доволі часто розробник програмного забезпечення стикається з проблемами такими як :

1. Передавання даних між різними модулями.
2. Повідомлення одного модуля про події, які відбулись в іншому модулі.
3. Виклик методів одного модуля з іншого.

Саме ці всі проблеми може вирішувати даний шаблон проектування. Делегування – основний шаблон проектування, в якому об’єкт зовнішньо виражає деяку поведінку, але в дийсності можна констатувати, що він передає відповідальність за здійснення цієї поведінки іншому звя’занному об’єктух[2]. Цей шаблон є фундаментальною абстракцією, на основі якої реалізовані інші шаблони, такі як абстракція[6].

В випадку даного додатку цей шаблон було застосовано для реалізації передачі відповідальності від модуля AnswerScreen модулю AnswersList (Рисунок 4.4). Перший — це модуль, в якому користувач дає відповідь на питання, а другий — це модуль, на якому користувач бачить повний список відповідей на конкретне питання.

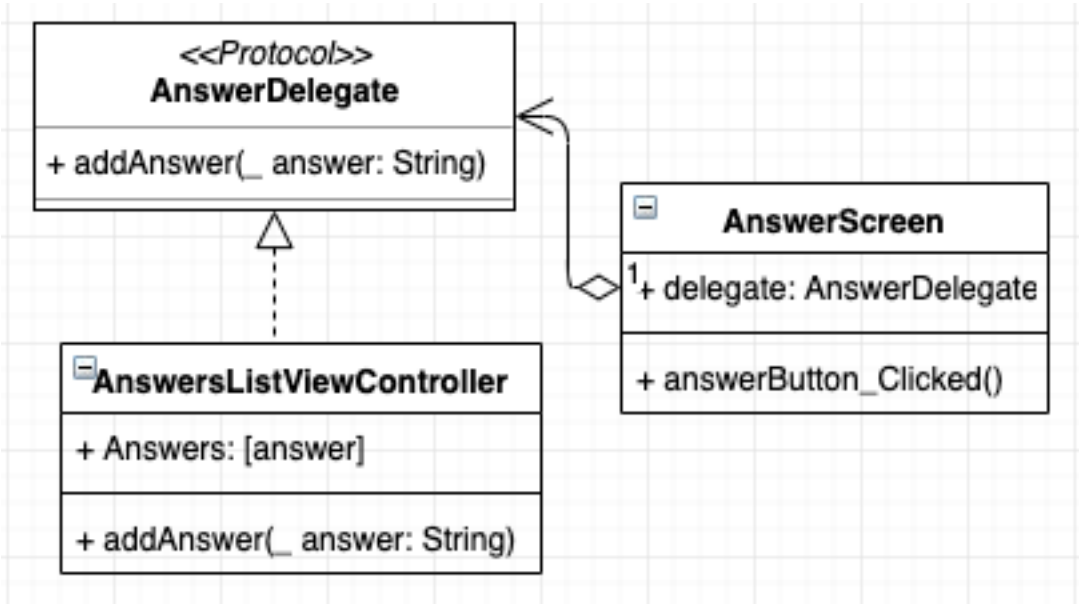

Рисунок 4.4 – Приклад делегування

Дуже важливо розуміти, що посилання на AnswerDelegate має бути слабким, для того щоб не утворювати циклу сильних посилань і потім ARC зміг очистити пам’ять.

Протокол AnswerDelegate може бути повторно використаним в будь-якому місці проекту. Наведений вище факт є досить важливим в побудові програмного забезпечення.

4.2. Робота з локальною базою даних

Як вже описувалося раніше, для роботи з локальною базою даних було обрано стандартний фреймворк від компанії “Apple” – CoreData, який вбудовано в платформу iOS.

Сутності в базі даних також було описано за допомогою конструктора інтерфейсів вбудованого в середу розробки Xcode.

Сутність предметів вимагає наявності унікального ідентифікатора, як і будь-який об’єкт в локальній базі даних. Також в даному об’єкті присутні такі поля, як ім’я, опис предмета та ідентифікатор викладача. Сутність предметів відмічена маркером @ManagedObject в коді програми, для того, що проінформувати компілятор, що

даний клас буде використовуватся фреймворком CoreData та буде вфкористовувати динамічну диспетчеризацію та середу виконання Objective-C.

Предмет (SubjectCD)	
Атрибут (Attribute)	Тип (Type)
id	String
name	String
descriptionSubject	String
teacherId	String

Таблиця 4.1 – Опис сутності предметів локальної бази даних

Сутність лекцій вимагає містити в собі атрибут, в якому буде зберігатися посилання на ідентифікатор предмета відповідної лекції. Також даний об’єкт містить в собі атрибути назви лекціє та її унікальний ідентифікатор.

Лекція (LectureCD)	
Атрибут (Attribute)	Тип (Type)
id	String
name	String
subject	String

Таблиця 4.2 – Опис сутності лекцій локальної бази даних

Для унікальних ідентифікаторів було обрано використовувати тип String, так як він займає менше пам’яті в базі ніж тип UUID. Але при записі об’єкта використовується UUID зведений до типу String. Дана операція не є дуже складною та затратною по ресурсам для операційної системи iOS.

4.2.1. Завантаження даних з локальної бази даних

При відсутності підключення до інтернету користувач має все одно отримати дані, тому в модулі зі списком предметів використовується завантаження даних з локальної бази при появі проблем з інтернетом. При реалізації методу для цього

спочатку формується запит, після чого з контексту повертається результат цього запиту і потім відбувається конвертація до звичних сутностей (Рисунок 4.5):

```
func loadDataFromCD() -> [Subject]? {
    //формування запиту до бази даних
    let request: NSFetchRequest<SubjectCD> = SubjectCD.fetchRequest()
    let sortDescriptor = NSSortDescriptor(key: "name", ascending: true)

    request.sortDescriptors = [sortDescriptor]

    let sources = try? context.fetch(request)
    var result: [Subject] = [Subject]()

    for item in sources! {
        //конвертація об'єкта та додавання до массиву
        result.append(Subject(object: item))
    }
    return result
}
```

Рисунок 4.5 – Реалізація методу завантаження списку предметів з бази даних

Такий підхід дозволяє користуватися додатком навіть без підключення до інтернету.

4.2.2. Збереження даних до локальної бази даних

При наявності підключення до інтернету, виконується запит на сервер, який повертає актуальні дані для конкретного користувача. Саме в цей момент потрібно обновити вже існуючі дані в базі даних та записати нові об’єкти, які було отримано з сервера (Рисунок 4.6).

Для роботи з базою даних використовувалися нові класи, такі як SubjectCD та LectureCD. Тобто для того, щоб зберегти дані в сховище потрібно спочатку конвертувати їх зі звичайнії об’єктів Subject або Lecture, які використовувалися для відображення, в сутності CoreData.

Даний факт було обумовлено тим, що об’єкти, які заходяться під впливом даної бібліотеки використовують динамічну диспетчеризацію, а це в свою чергу означає,

що операціє з ними будуть проводитися довше ніж зі звичайними об’єктами та класами, які в свою чергу використовують статичну диспетчеризацію.

```
func saveData(_ itemSource: [Subject]) -> Void {
    for item in itemSource {
        //створення нового або оновлення даних вже
            існуючого об'єкта
        let _ = try? SubjectCD.initOrFind(by: item)
        //збереження контексту
        try? context.save()
    }
}
```

Рисунок 4.6 – Реалізація методу збереження списку предметів

Даний метод демонструє простоту роботи з фреймворком CoreData. Доцільно робити записи до бази даних асинхронно або взагалі в новому потоці для того, щоб не блокувати головний потік виконання додатку, з яким має взаємодіяти користувач.

4.2.3. Використання UserDefaults

UserDefaults — є надзвичайно корисним для зберігання невеликих фрагментів даних у додатку для платформи iOS. Разом з цим, дане API являється дещо схожим на словник з даними.

В даному програмному застосунку використовується для того, щоб зберегти дані про поточного користувача (Рисунок 4.7):

```
private func saveLocal(_ user: User) {
    //створення об'єкта, який буде кодувати дані користувача
    let encoder = JSONEncoder()
    //кодування
    if let encoded = try? encoder.encode(user) {
        let defaults = UserDefaults.standard
        //додавання закодованих даних до UserDefaults за ключем
        defaults.set(encoded, forKey: "currentUser")
    }
}
```

Рисунок 4.7 – Реалізація методу збереження поточного користувача

Даний метод завжди викликається після проведення успішної авторизації та реєстрації.

Swift — дуже гнучка мова програмування, що передбачає створення своїх розширень для вже існуючих інструментів. Тому для запиту до UserDefaults про дані поточного користувача використовується розширення для UserDefaults, в якому знаходиться метод з самим запитом (Рисунок 4.8). Це рішення обумовлено частою потребою використання данних користувача.

```
//розширення UserDefaults для швидкого запита користувача
extension UserDefaults {
    func getUser() -> User? {
        let defaults = UserDefaults.standard
        //запит до UserDefaults за ключем
        if let savedUser = defaults.object(forKey: "currentUser") as? Data {
            //створення декодуючого об'єкта
            let decoder = JSONDecoder()
            //декодування
            if let loadedUser = try? decoder.decode(User.self, from: savedUser) {
                return loadedUser
            }
        }
        //якщо не вийшло декодувати об'єкт, то повертаємо nil
        return nil
    }
}
```

Рисунок 4.8 – Реалізація розширення для UserDefaults

Цей метод доступний з будь-якого файлу проекта. Даний факт обумовлений тим, що UserDefaults в мові програмування Swift, реалізовано, як Singleton.

4.3. Використання універсальних типів

Універсальні типи допомагають розробникам програмного забезпечення писати менше коду чим потрібно. Тобто, можна використовувати одні і ті ж самі методи, або класи для різних типів, які задовольняють умову відповідного універсального типу.

В даному додатку існує багато об’єктів, які ініціалізуються після запиту до віддаленої бази даних, тому було вирішено використати універсальний тип для класу ItemLoaderService, для того щоб не писати дублюючийся код (Рисунок 4.9).

```
class ItemLoaderService<T> where T: Firebaseable {
    //посиланя на базу даних
    var ref = Database.database().reference()

    func loadItems(folderName: String, complition: @escaping ([T]?) ->
        Void){
        //група користувача
        let group = UserDefaults.standard.getUser()!.group!
         //завантаження данних з FireBaseDatabase

            self.ref.child("groups").child(group).child(folderName)
            .observeSingleEvent(of: .value, with: { snapshot in
            //масив результатів
            var result = [T]()

            for child in snapshot.children {
                if let snapshot = child as? DataSnapshot,
                    //конвертація в об'єкти
                    let item = T.init(snapshot: snapshot) {
                    result.append(item)
                }
            }
            complition(result) //виклик функції завершення
        }) { error in
            complition(nil) //виклик функції заверення
        }
    }
}
```

Рисунок 4.9 – Реалізація класу завантаження даних з FirebaseDatabse за допомогою універсальних типів.

Екземпляр даного сервіса використовується в таких модулях, як списки предметів, лекцій, відповідей та навіть в поточній лекції. Для організації потокобезпечного сервіса, було обрано реалізувати звичайний клас, замість шаблону Singleton. Цей файкт обумовлено тим, що клас ItemLoaderService використовується в

багатьох модулях і можливий одночасний виклик методу цього класу з різних модулів програми.

Очевидно, що на даний універсальний тип накладається обмеження — він має реалізовувати протокол Firebaseable, який в свою чергу містить в собі ініціалізатор, що приймає параметр DataSnapshot (Рисунок 4.10). Цей параметр — це дані які повертає FirebaseDatabase після будь-якого запита.

```
protocol Firebaseable {
    init?(snapshot: DataSnapshot)
}
```

Рисунок 4.10 – Протокол Firebaseable

Такі класи, як Subject, Lecture, Answer, Question, задовольняють цей протокол, тобто реалізовують ініціалізатор з відповідним параметром. В середені даного метода відбувається конвертація DataSnapshot в відповідні поля сутності після чого створюється об’єкт.

Висновки до розділу 4

В даному розділі було розглянуто всі базові архітектурні та програмні рішення, які використовувалися в розроблюваному програмному забезпеченні. Також було продемонстровано роботу з локальною базою даних в додатку.

5. РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

У наступних розділах буде розглянута взяємодія користувача з програмним додатком, починаючи від звичайної істаляції. Також буде детально описано роботу основних модулів програмної системи.

5.1. Інсталяція та системні вимоги

Компанія “Apple” дуже ретельно ставиться до безпеки власних користувачів, через це завантажувати та встановлювати додатки, які не знаходяться в магазині “App Store”, у користувача – не вийде. Всі програми, які потрапили в маркет, обов'язково пройшли декілька етапів детальної первірки.

Даний додаток для iOS поки що не додавався в маркет, але в майбутньому це планується здійснити. Тому тільки після цього він буде доступний для кінцевих користувачів.

Для того щоб користувач міг завантажити додаток, йому потрібно мати iPhone або iPad з версією iOS не нижче 10.0, та бути зареєстрованим в “App Store”, також потрібно мати мінімум 50 мегабайтів вільної пам’яті на девайсі. Якщо попередні вимоги виконуються, то потрібно лише в пошукі від маркета знайти додаток за назвою “Lecture Time” і натиснути на кнопку “Встановити”.

5.2. Заставка та авторизація, реєстрація користувача

Після того, як користувач завантажить та відкриє даний програмний продукт він одразу побачить заставку (Рисунок 5.1). Додатки для платформи iOS мають властивість під час запуска завантажувати деякі з файлів проекта в оперативну пам’ять і відповідна операція займає певний час. За попередніми даними в наступній версії операційної системи планується збільшити швидкість запуску додатку на 30%.

Саме цей факт обумовлює високу швидкодію програмних застосунків для даної операційної системи.

Спираючись на попередньо вказаний факт було вирішено реалізувати заставку для застосунка(Splash Screen), для того, щоб користувач не звертав уваги на деяку затримку під час запуска додатка. І тому він побачить екран з анімацією, в якій будуть присутні назва прогромної системи та її логотип. Ці два елемента будуть анімовані, а саме: вони будут зблішуватися за розміром на протязі всього завантаження додатка.

Рисунок 5.1 – Екран заставки під час завантаження даних для програмного застосунка

Так як, цей додаток являється демонстраційною версією і розроблений, поки що на основі групи ТІ-51. Тому додаток передбачає авторизацію та реєстрацію студентів цієї групи або викладачів, які викладають предмети в ній.

Після того як користувач побачить заставку, його буде одразу перенаправлено на сторінку авторизації та реєстрації. Переключатися між ними можна за допомогою простого свайпу в будь-яку сторону, так як наявно лише дві сторінки.

Сторінка авторизації оформлена за стилем дизайну Flat UI[1] (Рисунок 5.2). Авторизація доступна, як для студентів, так і для викладачів.

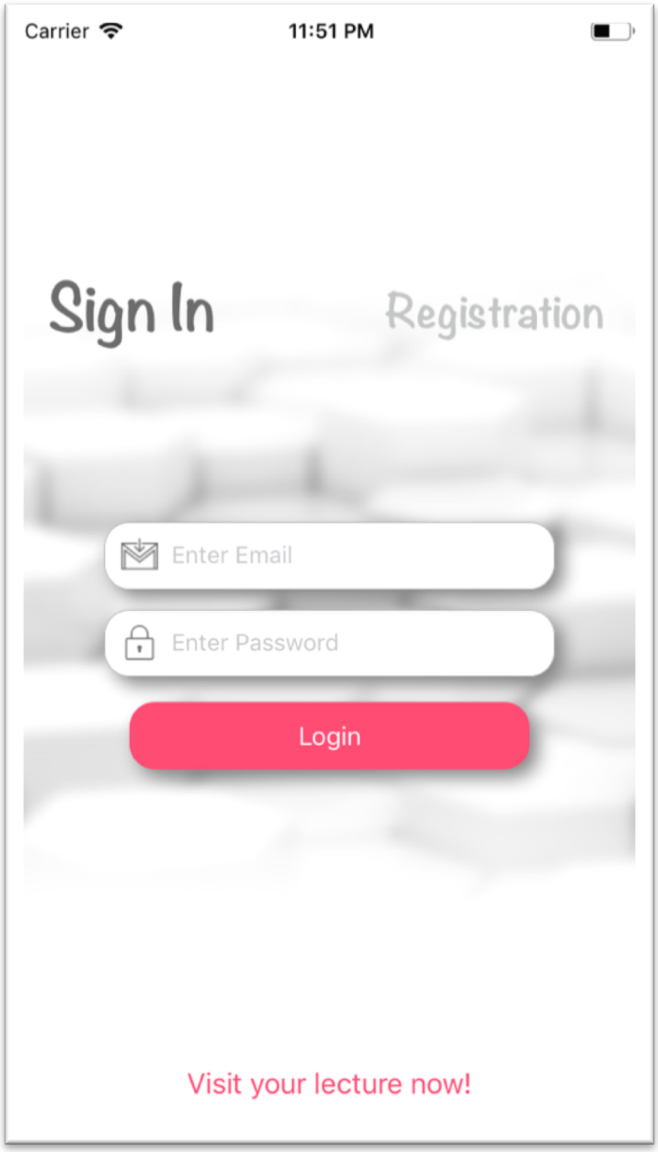

Рисунок 5.2 – Екран авторизації

На даному екрані знаходиться форма для входу користувача в головний модуль програми. Також тут пристуня валідація для полів. Якщо ввести невірні дані або залишити поля пустими і натиснути кнопку “Login” на екрані з’явиться повідомлення про помилку у вигляді стандартного попередження iOS.

Дуже важливо розуміти, що не пройшовши тап авторизації, користувач не зможе перейти до основних модулів програмного продукту. Це зроблено для того, щоб забезпечити безпечне користування даним додатком.

Якщо користувач ще не створив свій особистий акаунт, то він може це здійснити на екрані реєстрації (Рисунок 5.3).

При цьому після того, як акаунт успішно буде зараєстровано, то користувач одразу перенаправиться в головний модуль додатку. Тобто, йому не потрібно буде потім повертатися на сторінку авторизації і вводити свою електронну адрсу та пароль для входу ще раз.

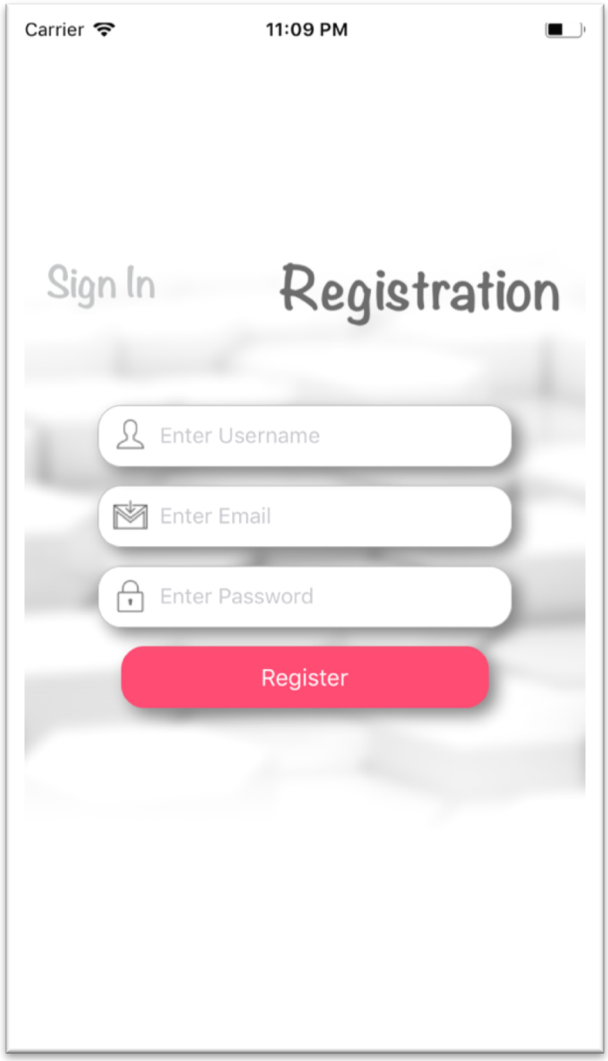

Рисунок 5.3 – Екран реєстрації

Для проведення реєстрації користувачу необхідно ввести прізвище та ім’я в поле “Username”, та заповнити поля з електронною адресою та паролем, кількість

символів якого більша від п’яти. Якщо дана електронна адреса вже використовується, то з'явиться повідомлення про помилку і створення нового акаунта не відбудеться. Повідомлення про помилку буде з’являтися кожен раз, коли користувач буде вводити не коректну електронну адресу.

5.3. Основні модулі додатку

Після успішної авторизації користувач потрапляє на навігаційну сторінку, де йому доступні переходи екрани з поточною лекцією, списком предметів або профілем.

Дана сторінка є різновидом UITabBarController, яка була створена за допомогою будівника інтерфейсів в файлі tabs.storyboard.

5.3.1. Модуль поточної лекції

По замовчуванні першим відкривається модуль з поточною лекцією, так як основні події відбуваються саме в ньому.

На цьому екрані користувач бачить всі питання щодо поточної лекції та також може задати питання, які йому цікаві. Питання мають вигляд стандартних повідомлень на платформі iOS.

Також даний екран використовується для екранів з питаннями для попередніх лекцій. На ці екрани можна перейти за допомогою другої кнопки в навігаційному меню – “Subjects”. Після чого обравши потрібний предмет, а потім лекцію користувач побачить подібну сторінку до сторінки поточної лекції.

Всі питання є анонімними, що є досить важливим, так як студент не буде перейматись та ніяковіти, якщо він поставить очевидне запитання.

Даний екран являється найважливішим в додатку, так як саме в ньому відбувається живий обіг питань під час лекції. Можна вважати, що саме в цьому місці і розв’язується одна з поставлених задач.

Тобто студент під час лекції просто має написати в форму знизу – запитання, яке його турбує, і в нього не буде ніякої необхідності питати лектора наживо в аудиторії або під час дистанційного навчання за допомогою інтернет конференції (Рисунок 5.4).

Рисунок 5.4 – Екран поточної лекції

Якщо залишити поле для питання пустим і натиснути на кнопку “Send” – пусте питання не буде відправлене в базу даних.

З даного модуля при натисканні на одне із запитань відбувається перехід на екран відповідей на обране питання (Рисунок 5.5).

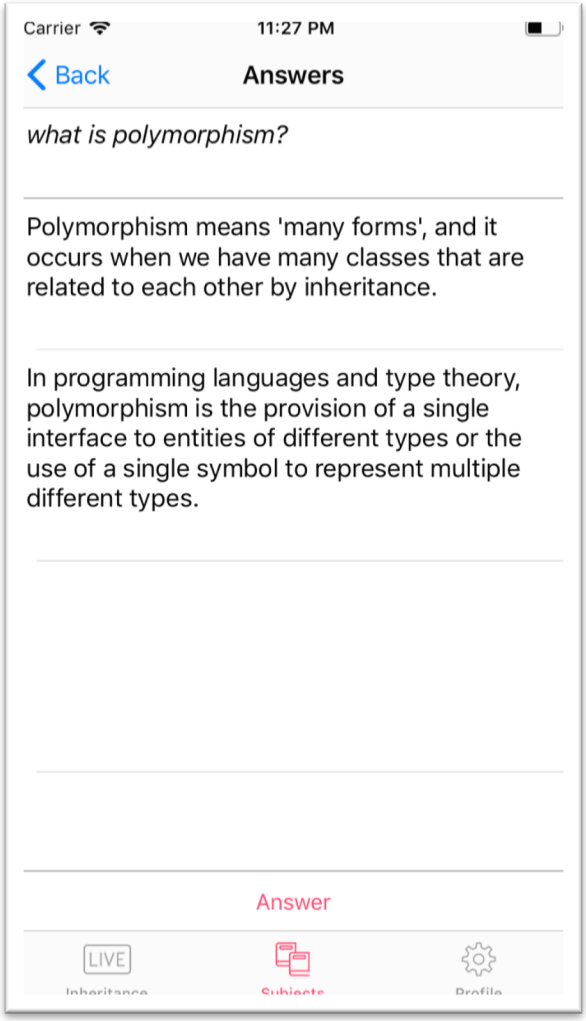

Рисунок 5.5 – Екран відповідей на запитання

Дана сторінка містить в собі список з відповідями на поставлене питання. З цього екрану можна повернутися назад до списку питань або ж за допомогою кнопок навігації перейти на екран зі списком предметів або власного профіля користувача програмної системи.

Першим в цій таблиці знаходиться поставлене запитання. Прокрутивши таблицю, то її перша комірка залишиться на своїй позиції. Це досить важливо, щоб користувач завжди бачив перед собою поставлене питання, а не тільки відповіді на нього.

Знизу знаходиться кнопка “Answer”, яка також при прокручуванні залишається на своєї позиції, щоб користувач в будь-який момент міг вільно перейти на екран додавання нової відповіді на поставлене питання.

Ця сторінка дещо відрізняється для користувачів групи “Викладач” та “Студент”. Це обумовлено тим, що студенти можуть тільки переглядати відповіді дані на поставлене запитання. Вкладач же в свою чергу, має змогу ще й відмічати відповідь, як вірну та редагувати її (Рисунок 5.6).

Дані функції для користувача групи “Викладачі” було реалізовано для здійснення кращого UX[8] в додатках для iOS. Тому викладачу не потрібно буде передруковувати частково вірну відповідь на питання з додаванням власних думок, а йому лише буде необхідно відредагувати близьку до вірної відповідь і помітити її як вірну, якщо це необхідно.

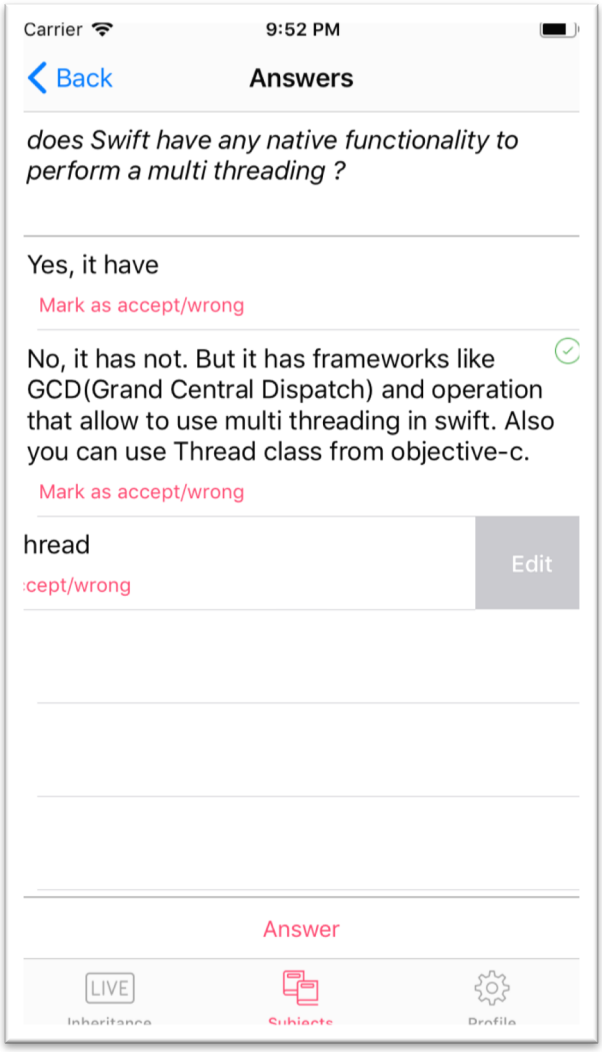

Рисунок 5.6 – Екран відповідей на запитання для користувача групи “Викладачі”

Якщо користувач хоче дати відповідь на запитання, то натиснувши кнопку “Answer” він перейде на новий модуль, де він зможе дати розгорнуту відповідь (Рисунок 5.7).

Дана сторінка є однією з основних сторінок додатка, яка допомогає користувачам обох груп (“Студент” та “Викладач”) здійснювати комунікацію між собою.

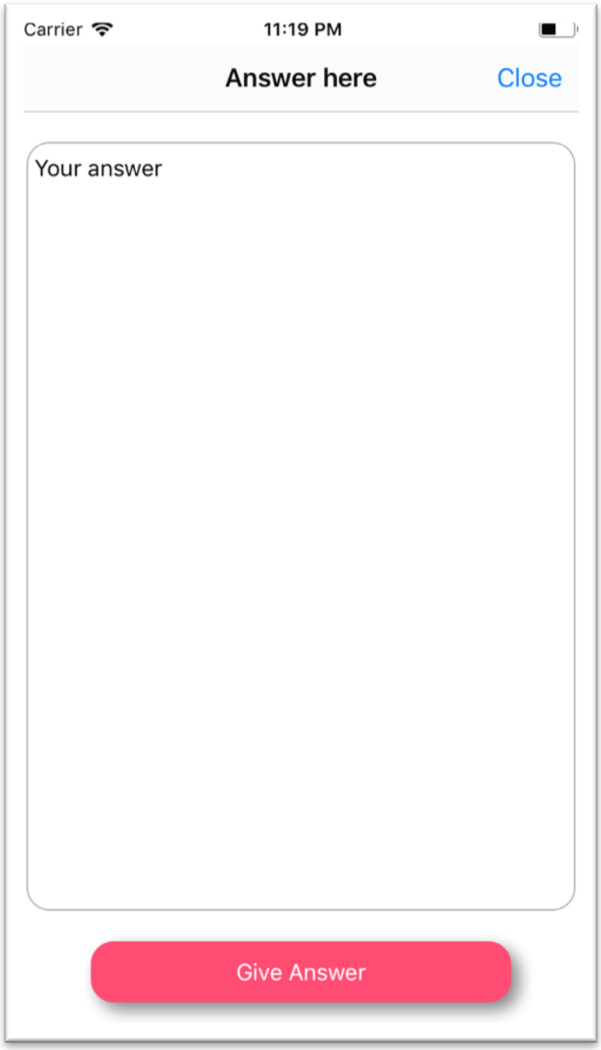

Рисунок 5.7 – Екран додавання нової відповіді на запитання

При натисненні на кнопку “Give Answer” текст, який знаходитиметься в полі для відповіді відправиться на сервер та буде доданий до списку. Так як ця сторінка є модальною, на ній передбачена кнопка “Close”, щоб повернутися назад, якщо користувач передумає додавати нову відповідь.

5.3.2. Модуль предметів та попередніх лекцій

Якщо користувач натисне на кнопку “Subjects” в навігаційному меню, то його буде перенаправлено на екран зі списком предметів (Рисунок 5.8).

Даний екран використовується частіше під час роботи студента вдома для аналізу пройденого на лекції, або підчас перегляду викладачем лекцій, які завершились для того, шоб відповісти на всі поставлені запитання.

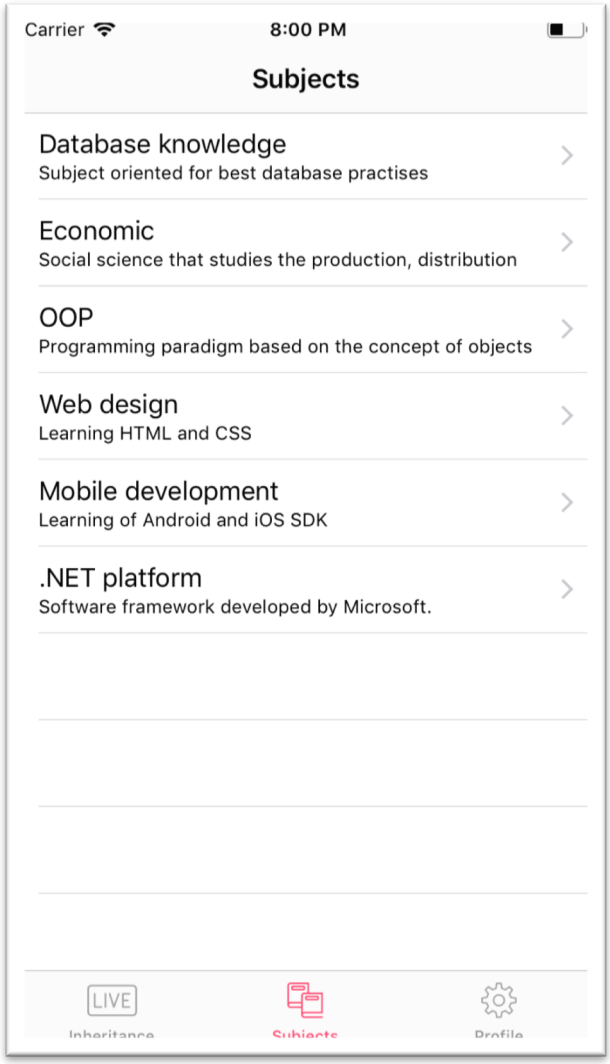

Рисунок 5.8 – Екран списку предметів користувача

На цій сторінці користувач бачить список зі своїми предметами та їх описом. При виборі одного з предметів відбувається перехід на екран зі списком лекцій. При Виборі лекції, яка вже завершилась – користувач не може додати нове запитання.

Якщо наявне підключення до інтернету, то користувач отримає нові дані, але якщо інтернет буде відсутнім, то користувач отримає ті дані, які були завантажені минулого раза.

Саме цю можливість забезпечує фреймворк CoreData.

5.3.3 Модуль профіля користувача

Якщо користувач натисне на кнопку “Profile” в навігаційному меню, то його буде перенаправлено на екран з власним акаунтом (Рисунок 5.9) .

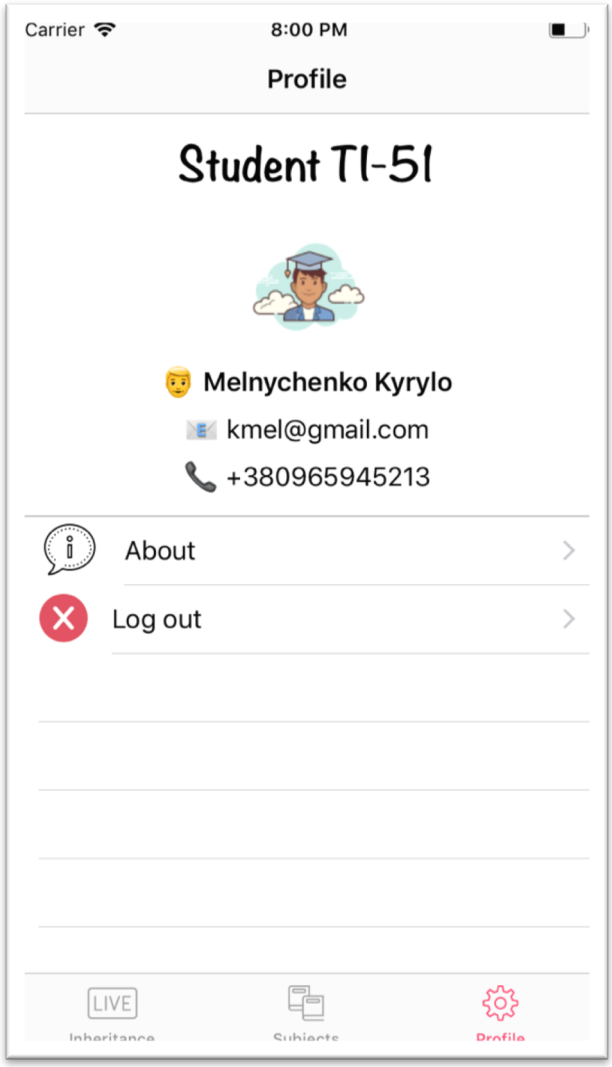

Рисунок 5.9 – Екран профілю користувача

На цій сторінці знаходиться інформація та деякі дані користувача, які було заповнено під час реєстрації. Також доступні дві дії – вихід з акаунта та перегляд

сторінки “About”. Якщо користувач натисне кнопку “Log Out” він буде перенаправлений на екран авторизації.

Модуль “About” в собі містить іконку та назву даної системи, як і заставка при завантаженні додатка (Рисунок 5.10). Головними тезами цього екрану являється те, щоб донести до користувачів, що саме вирішує система і що саме планується зробити в майбутньому для того, щоб покращити та надати змогу автоматизувати процес проведення занять в університетах, школах та на приватних курсах.

Також ставиться за мету оповістити користувачів, що дана версія додатку поки що є демонстраційною, тобто вся функціональність та всі дані використовуються тільки для однієї групи ТІ – 51.

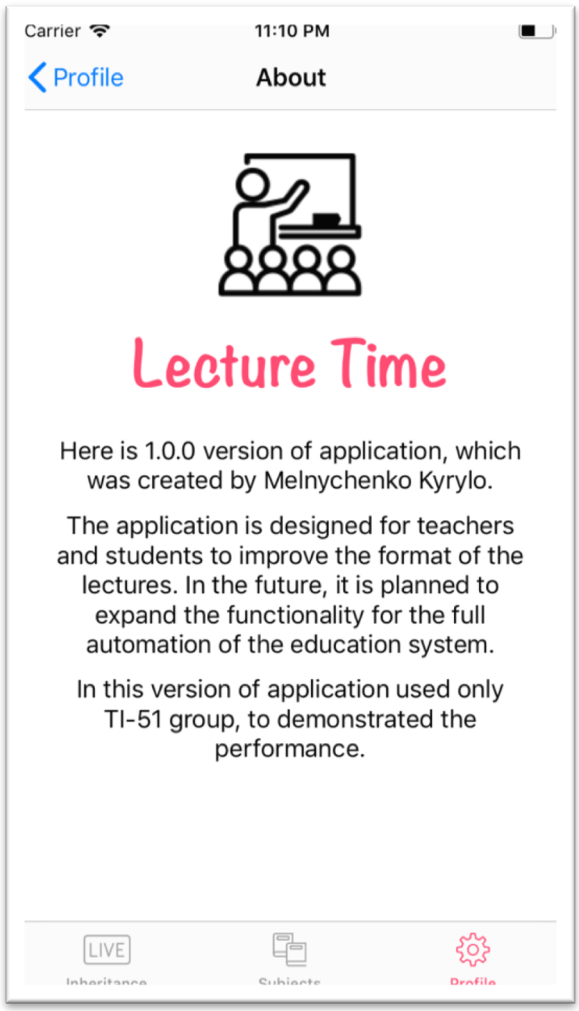

Рисунок 5.10 – Екран модуля “About”

На цій сторінці користувач бачить інформацію про додаток, ким він написаний, його призначення та версію.

З цього екрану користувач може повернутися назад до вікна профілю користувача натиснувши кнопку “Profile”. Також йому доступні кнопки нижнього меню для переходу між іншими основними модулями додатка.

Висновки до розділу 5

В даному розділі було розглянуто процес інсталяції додатка, взаємодію користувача з програмною системою. Також цей розділ ілюструє простоту використання продукту для всіх груп користувачів (“Викладач” та “Студент”).

ВИСНОВКИ

Результатом виконання дипломної роботи та проходження практики стала розробка додатку для платформи iOS. Програмний продукт розроблений на мові програмування Swift в середовищі розробки Xcode.

Було виконано дослідження існуючих аналогічних або близьких за змістом програми для систематизації проведення лекцій, проаналізовані їх недоліки та запропоновані інтерфейсні та функціональні рішення для покращення зручності користування користувачів.

Розроблений додаток надає змоги:

1. користувачам задавати питання під час лекції, не відволікаючи при цьому інших студентів;
2. використовувати додаток анонімно;
3. переглядати список лекцій та предметів;
4. відповідати на поставлені питання;
5. переглядати питання та відповіді по попередніх лекціях;
6. переглядати власний профіль;
7. використовувати певні модулі навіть без підключення до інтернету;

Також в програмі наявні модулі авторизації, реєстрації та модуль, в якому знаходяться факти про розроблювану систему. Було передбачено дві групи користувачів (“Викладач” та “Студент”). Групу викладач було наділено спеціальними можливостями:

— Відмічати відповідь, як вірну.

— Редагувати відповідь.

За допомогою архітектурних шаблонів було створено легко розширювану систему, за допомогою якої в майбутньому можливо буде повністю автоматизувати та систематизувати учбовий процес.

Також було розроблено гнучкий та адаптивний інтерфейс для даної програмної системи. Цей факт забезпечує можливість використовувати застосунок на мобільних

телефонах iPhone та планшетах IPad з відповідними операційними системами — iOS та iPadOs.

Було використано різноманітні засоби середи розробки Xcode та конструктора інтерфейса за допомогою, яких розроблено дружній інтерфейс для використання користувачами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. «Плоский дизайн»: с чего начать? Пять основных принципов Flat дизайна [Електронний ресурс] // powerbranding – Режим доступу: http://powerbranding.ru/design/flat-design-june13/.

2.Приёмы объектно-ориентированного проектирования. Паттерны проектирования / Э.Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. – Харьков: Питер, 2001. – 368 с.

3. Бодягін І. Model-View-Controller / Іван Бодягін. // RSDN Magazine. – 2016. – №2.

4. The book of VIPER [Електронний ресурс]. / Rambler & Co // 2014. – 83 с. – Режим доступу: https://github.com/strongself/The-Book-of-VIPER

5. Нейбургскі М. IOS 10 програмування за допомогою Swift, Xcode / Нейбургскі Метт. // O'Reilly Media, – 2016. – 620 с.

6. Чистая архитектура. Искусство разработки программного обеспечения / Роберт Мартин, – Санкт-Питербург: Питер, 2016. – 352 с.

7. Swift: разработка приложений в среде Xcode для iPhone и iPad с использованием iOS SDK / Дэвид Марк, Джек Наттинг, Ким Топли, Фредрик Т. Олссон, Джефф Ламарш // Диалектика-Вильямс. – 2016. – 816 с.

8. Фаулер М. GUI Architectures / Мартін Фаулер. // All Sections ThoughtWorks. – 2006.

9. Василий Усов: Swift. Основы разработки приложений под iOS и macOS / Василий Усов // Издательство Питер. – 2018. – 448 с.

10. iPhone SDK. Разработка приложений / Джонатан Здзиарски // BHV-СПб. – 2010. – 512 с.

11. Core Data by Tutorials: iOS 12 and Swift 4.2 Third edition / Ray Wenderlich, Aaron Douglas, Saul Mora // Razeware LLC. – 2018. – 280 с.

12. Learning Core Data for iOS: A Hands-On Guide to Building Core Data Applications / Tim Roadley // Addison-Wesley Professiona. – 2013. – 472 с.

13. Руководство Swift [Електронний ресурс] // swiftbook – Режим доступу: https://swiftbook.ru/content/languageguide/

14. iOS Drawing: Practical UIKit Solutions / Erica Sadun // CreateSpace Independent Publishing Platform. – 2014. – 308 с.

15. Foundation Swift / Alex Halloohan, William Irving, Dave Shark, Bill Spencer, Lumbo Thevathasan // Apress. – 2014. – 123 с.

16. The Object-Oriented Thought Process (4th Edition) / Matt A. / Addison-Wesley Professional. – 2013. – 336 с.

ДОДАТОК А

Система поліпшення проведення занять (клієнт для iOS)

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ51183_19Б

Аркушів 2

Київ 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ51183_19Б	Записка_Мельниченко.docx	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ51183_19Б	AcademicPerformanceApp.app	Основний компонент додатку

ДОДАТОК Б

Система поліпшення проведення занять (клієнт для iOS)

Текст програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ51183_19Б

Аркушів 9

Київ 2019

Текст програми. Модуль авторизації та реєстрації

```
import UIKit

class LoginViewController: UIViewController, LoginViewProtocol, ViperModuleTransitionHandler {
    var presenter: LoginPresenterProtocol!
    let configurator: LoginConfiguratorProtocol = LoginConfigurator()

    weak var parentController: UIViewController?

    //MARK: - Otuletss
    @IBOutlet weak var loginTextField: UITextField!
    @IBOutlet weak var passwordTextField: UITextField!
    @IBOutlet weak var loginButton: UIButton!
    @IBOutlet weak var forgetPasswordLabel: UILabel!

    @IBOutlet weak var activityIndicator: UIActivityIndicatorView!
    //MARK: - Life cycle
    override func viewDidLoad() {
        configurator.configure(with: self)
        setUpUI()
    }

    override func viewDidAppear(_ animated: Bool){
        loginButton.isHidden = false
        loginTextField.isHidden = false
        passwordTextField.isHidden = false
    }

    override func viewDidDisappear(_ animated: Bool) {
        super.viewDidDisappear(animated)

        loginTextField.text = ""
        passwordTextField.text = ""
    }

    //MARK: - UI setups
    func setUpUI() {
        let gesture = UITapGestureRecognizer(target: self, action: #selector(forgetPasswordClicked(_:)))
        forgetPasswordLabel.isUserInteractionEnabled = true
        forgetPasswordLabel.addGestureRecognizer(gesture)

        loginButton.layer.cornerRadius = MAIN_CORNER_RADIUS
        setShadow(for: loginButton)

        loginTextField.setPaddingWithImage(imageName: "emailIcon")
        setBorder(for: loginTextField)
        setShadow(for: loginTextField)

        passwordTextField.setPaddingWithImage(imageName: "passwordIcon")
        setBorder(for: passwordTextField)
        setShadow(for: passwordTextField)
    }

    @objc func forgetPasswordClicked(_ sender: Any) {
    }

    @IBAction func loginButtonClicked(_ sender: Any) {
        activityIndicator.startAnimating()
        performLogin(username: loginTextField.text, password: passwordTextField.text)
```

```
    }

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        activityIndicator.stopAnimating()

        let configurationHolder = segue.destination as? ForgetPasswordViewController
        configurationHolder?.CreateModule()
        (sender as? SegueInfo)?.configurationBlock?(configurationHolder?.presenter)
    }

    //MARK: - Implementation of LoginViewProtocol
    func configureView(with username: String){

        if (!loginTextField.isHidden) { return }

        UIView.animate(withDuration: 0.5, delay: 0, options: .curveEaseInOut,
                       animations: {
                        self.loginButton.alpha = 1
                        self.loginTextField.alpha = 1
                        self.passwordTextField.alpha = 1
                        self.forgetPasswordLabel.alpha = 1
        },
                       completion: { _ in self.loginButton.isHidden = false
                        self.loginTextField.isHidden = false
                        self.passwordTextField.isHidden = false
                        self.forgetPasswordLabel.isHidden = false
        })

        parentController?.viewWillAppear(true)
    }

    func performLogin(username: String?, password: String?) {
        presenter.performLogin(username: username, password: password)
    }

    func showError(error: String) {
        activityIndicator.stopAnimating()
        let alert = UIAlertController(title: "Error", message: error, preferredStyle: .alert)
        alert.addAction(UIAlertAction(title: "OK", style: UIAlertAction.Style.default, handler: nil))
        self.present(alert, animated: true, completion: nil)
    }

            }


protocol LoginViewProtocol: class {
    func configureView(with username: String) -> Void

    func performLogin(username: String?, password: String?) -> Void

    func showError(error: String) -> Void
            }


protocol LoginPresenterProtocol: class {
    var router: LoginRouterProtocol! { set get }

    func configureView() -> Void
    //View to presenter, presenter to interactor
    func performLogin(username: String?, password: String?) -> Void
    //interactor to presenter, presenter to view
    func loginDidComplete() -> Void
```

```
    func loginDidFail(with error: Error) -> Void

    func moveToFP() -> Void
}

protocol OutsideNotifierProtocol: class {
    func notifyAboutAppearing() -> Void
    func notifyAboutDisapearing() -> Void
}




class LoginPresenter: LoginPresenterProtocol {
    weak var viewController: LoginViewProtocol!
    var router: LoginRouterProtocol!
    var interactor: LoginInteractorProtocol!

    required init(viewController: LoginViewProtocol) {
        self.viewController = viewController
    }

    func configureView() {
        viewController.configureView(with: "")
    }

    func performLogin(username: String?, password: String?) {
        interactor.performLogin(username: username ?? "", password: password ?? "")
    }

    func loginDidComplete() {
        router.moveInsideApp()
    }

    func loginDidFail(with error: Error) {
        viewController.showError(error: error.localizedDescription)
    }

    func moveToFP() {
        router.moveToForgotPasswordPage(del: self as OutsideNotifierProtocol)
    }
}

extension LoginPresenter: OutsideNotifierProtocol{
    func notifyAboutAppearing() {
        self.configureView()
    }

    func notifyAboutDisapearing() {
    }
}



class LoginInteractor: LoginInteractorProtocol {
    weak var presenter: LoginPresenterProtocol!
    private let validationService: ValidationServiceProtocol! = ValidationService()
    private let authorizationService: AuthServiceProtocol! = RemoteAuthService()
    private let authFB = AuthorizationService()

    init(presenter: LoginPresenterProtocol){
        self.presenter = presenter
    }
```

```
    func performLogin(username: String, password: String) {
        guard (validationService.validate(for: username)), (validationService.validate(for: password)) else {
            presenter.loginDidFail(with: validationService.error!)
            return
        }
        authFB.perform(username, password) { error in
            if let error = error {
                self.presenter.loginDidFail(with: error)
            } else {
                self.presenter.loginDidComplete()
            }

        }

    }
            }


class LoginConfigurator: LoginConfiguratorProtocol {
    func configure(with viewController: LoginViewProtocol) {
        let viewControllerUnwrap = viewController as? LoginViewController
        let presenter = LoginPresenter(viewController: viewControllerUnwrap!)
        let interactor = LoginInteractor(presenter: presenter)
        let router = LoginRouter(viewController: viewControllerUnwrap!)

        viewControllerUnwrap?.presenter = presenter
        presenter.interactor = interactor
        presenter.router = router
    }
            }


class RegistrationViewController: UIViewController, RegistrationViewProtocol, ViperModuleTransitionHandler {

    let configurator: RegistrationConfiguratorProtocol! = RegistrationConfigurator()
    var presenter: RegistrationPresenterProtocol!

    //MARK: - Outlets
    @IBOutlet weak var phoneTextField: UITextField!
    @IBOutlet weak var usernameTextField: UITextField!
    @IBOutlet weak var emailTextField: UITextField!
    @IBOutlet weak var passwordTextField: UITextField!
    @IBOutlet weak var registerButton: UIButton!

    @IBOutlet weak var activityIndicator: UIActivityIndicatorView!
    override func viewDidLoad() {
        configurator.configure(with: self)
        setUpUI()
    }

    override func viewDidDisappear(_ animated: Bool) {
        super.viewDidDisappear(animated)

        emailTextField.text = ""
        passwordTextField.text = ""
        usernameTextField.text = ""
        phoneTextField.text = ""
    }

    func setUpUI() {
        registerButton.layer.cornerRadius = MAIN_CORNER_RADIUS
        setShadow(for: registerButton)
```

```
        usernameTextField.setPaddingWithImage(imageName: "loginIcon")
        setBorder(for: usernameTextField)
        setShadow(for: usernameTextField)

        emailTextField.setPaddingWithImage(imageName: "emailIcon")
        setBorder(for: emailTextField)
        setShadow(for: emailTextField)

        phoneTextField.setPaddingWithImage(imageName: "phoneNumber")
        setBorder(for: phoneTextField)
        setShadow(for: phoneTextField)

        passwordTextField.setPaddingWithImage(imageName: "passwordIcon")
        setBorder(for: passwordTextField)
        setShadow(for: passwordTextField)
    }



    @IBAction func registerButtonClicked(_ sender: Any) {
        activityIndicator.startAnimating()
        performRegistration(username: usernameTextField.text, email: emailTextField.text, password: passwordTextField.text,
phoneNumber: phoneTextField.text)
    }

    func configureView(with username: String, email: String) {

    }

    func performRegistration(username: String?, email: String?, password: String?, phoneNumber: String?) {
        presenter.performRegistration(username: username, email: email, password: password, phoneNumber: phoneNumber)
    }

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        activityIndicator.stopAnimating()
    }

    func showError(error: String) {
        activityIndicator.stopAnimating()
        let alert = UIAlertController(title: "Error", message: error, preferredStyle: .alert)
        alert.addAction(UIAlertAction(title: "OK", style: UIAlertAction.Style.default, handler: nil))
        self.present(alert, animated: true, completion: nil)
    }
}
```

Основні модулі програми

```
import UIKit

protocol AnswerDelegate: class {
    func addAnswer(_ answer: String) -> Void
}

class AnswersListviewController: UIViewController {

    @IBOutlet weak var tableView: UITableView!
    @IBOutlet weak var questionLabel: UILabel!

    var answers = [Answer]()
```

```
    let loader = ItemLoaderService<Answer>()
    let builder = EntityBuilder()

    var currentQuestion: Question?
    var position: Int!
    var id: String!
    var isAppend: Bool!

    override func viewDidLoad() {
        super.viewDidLoad()

        self.tableView.delegate = self
        self.tableView.dataSource = self
        //self.tableView.allowsSelection = false
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        answers = [Answer]()

        guard Connectivity.isConnectedToInternet() else {
            self.displayAlert(title: "Error.", message: "Can't load questions. Check your internet connection.")
            return
        }

        if let question = currentQuestion {
            loader.loadItems(folderName: "answers") { result in
                for item in result! {
                    if item.questionId! == self.currentQuestion?.id! {
                        self.answers.append(item)
                    }
                }

                if (self.answers.count > 0) {
                    self.tableView.reloadData()
                }
            }
        }

        self.title = "Answers"
        self.questionLabel?.text = currentQuestion?.text!
    }

    override func viewDidDisappear(_ animated: Bool) {
        super.viewDidDisappear(animated)
    }

    @IBAction func answerButton_Clicked(_ sender: Any) {
        self.id = currentQuestion!.id! + "\(answers.count)"
        self.isAppend = true
        performSegue(withIdentifier: "segueToModal", sender: nil)
    }

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        let vc = segue.destination as! AnswerScreenViewController

        vc.answerDelegate = self
        vc.answerText = isAppend ? nil : answers[position].text!
    }
}

extension AnswersListviewController: AnswerDelegate {
```

```
    func addAnswer(_ answer: String) {
        //let id = currentQuestion!.id! + "\(answers.count)"
        let answer = Answer(id: id, text: answer, questionId: self.currentQuestion!.id!, accepted:  false)
        //answers.append(answer)
        //tableView.reloadData()

        let newValue = ["id": id, "text": answer.text!, "questionId": answer.questionId!, "accepted": false] as [String : Any]

        builder.build(folder: "answers", value: newValue, id: id)
    }
}

extension AnswersListviewController: UITableViewDelegate, UITableViewDataSource {
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return answers.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "AnswerCell") as! AnswerCell
        cell.answer = answers[indexPath.row]
        cell.answerLabel.text = answers[indexPath.row].text ?? ""
        cell.acceptImage.image = UIImage(named: "accept")
        cell.acceptImage.isHidden = !answers[indexPath.row].accepted!
        cell.AcceptButton.isHidden = !UserDefaults.standard.getUser()!.isTeacher!
        cell.likeImage.image = UIImage(named:"like")
        cell.viewForStudent.isHidden = true
        cell.liked = false

        return cell
    }

    func tableView(_ tableView: UITableView, editActionsForRowAt indexPath: IndexPath) -> [UITableViewRowAction]?
    {
        let update = UITableViewRowAction(style: .normal, title: "Edit") { action, index in
            self.position = index.row
            self.id = self.answers[self.position].id!
            self.isAppend = false
            self.performSegue(withIdentifier: "segueToModal", sender: nil)
        }
        return [update]
    }

    func tableView(_ tableView: UITableView, canEditRowAt indexPath: IndexPath) -> Bool {
        return UserDefaults.standard.getUser()!.isTeacher ?? false
    }
}

import Foundation
import FirebaseDatabase

protocol Firebaseable {
    init?(snapshot: DataSnapshot)
}

class ItemLoaderService<T> where T: Firebaseable {
    //посилання на базу даних
    var ref = Database.database().reference()

    func loadItems(folderName: String, complition: @escaping ([T]?) -> Void){
        //група користувача
        let group = UserDefaults.standard.getUser()?.group ?? "ti-51"
         //завантаження данних з FireBaseDatabase
        self.ref.child("groups").child(group).child(folderName).observeSingleEvent(of: .value, with: { snapshot in
```

```
            //масив результатів
            var result = [T]()

            for child in snapshot.children {
                if let snapshot = child as? DataSnapshot,
                    //конвертація в об'єкти
                    let item = T.init(snapshot: snapshot) {
                    result.append(item)
                }
            }
            complition(result) //виклик функції завершення
        }) { error in
            complition(nil) //виклик функції заверення
        }
    }
}

class EntityBuilder {
    var ref = Database.database().reference()

    func build(folder: String, value: [String: Any], id: String) {
        let group = UserDefaults.standard.getUser()!.group!
    self.ref.child("groups").child(group).child(folder).child(id).setValue(value)
    }
}
import UIKit

class LectureListViewController: UIViewController, LectureListViewProtocol, ViperModuleTransitionHandler {
    @IBOutlet weak var tableView: UITableView!

    let configurator: LectureListConfiguratorProtocol = LectureListConfigurator()

    var presenter: LectureListPresenterProtocol!

    var subject: Subject?

    override func viewDidLoad() {
        super.viewDidLoad()

        tableView.delegate = self
        tableView.dataSource = self

        configurator.configure(with: self)
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        self.title = subject?.name

        presenter.setUpViewWithData(subject)
    }

    override func viewDidDisappear(_ animated: Bool) {
        super.viewDidDisappear(animated)

        presenter.setUpViewWithData(nil)
    }

    func reloadData() {
        tableView.reloadData()
    }
```

ДОДАТОК В

Технологія взаємодії застосунків MATLAB і C# у комплексі моделювання гідроакустичних процесів

Опис програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ51183_19Б

Аркушів 9

Київ 2019

АНОТАЦІЯ

Даний додаток містить опис програмної системи, розробленої для покращення проведення занять. Відповідна програмна система є застосунком для платформи iOS. Створена система виконує такі завдання:

— надавання змоги користувачам задавати питання під час лекцій;

— перегляд попереднього матеріалу;

— забезпечення анонімності користувачів;

— наділення певних груп користувачів певними спеціальними можливостями.

При розробці програмної системи використовувалась мова Swift. Для розробки було обрано середу розробки Xcode десятої версії. В додатку використані: інструмент для роботи з локальною базою даних – CoreData; базові iOS бібліотеки – UIKit та Foundation; засоби Firebase.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку міститься опис програмної системи покращення проведення занять (клієнт iOS) . У додатку Б міститься програмний код головних модулів розробленого програмного застосунку.

Даний додаток працює в операційних системах Apple — iOS та iPadOS.

При розробці використовувалась мова програмування Swift у середовищі розробки Xcode. Також було використано такі бібліотеки, як UIKit, Foundation, CoreData, Firebase.

.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблений програмний застосунок поліпшує проведення занять в університетах, школах та на приватних курсах. А саме додаток вирішує проблеми комунікації слухачів та викладача під час лекції.

Функціональні обмеження на використання додатку передбачено тільки групою користувача в системі.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Для реалізації програмного застосунка знадобилось використати сервер на базі Firebase, для того, щоб синхронізувати дані для кожного користувача.

Підключення до віддаленої бази даних відбувається в класах LiveObserving та ItemLoaderService, який використовує універсальний тип, для доступу з кожного модуля.

Повідомлення на екрані поточної лекції використовує LiveObserving для того, щоб відстежити нові дані на сервері.

Під час кожного запиту перевіряється наявні дані в локальній базі даних, та якщо вони вже не актуальні — перезаписуються нові дані.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для забезпечення повноцінної роботи та досягнення високої ефективності роботи створеного програмного застосунку було використано мову програмувння Swift та середу розробки Xcode.

В розроблюваному додатку використовуюсь такі бібліотеки, як FirebaseDatabase, Core Data.

Даний програмний застосунок може використовуватсь на платформах iOS та iPadOS.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Розроблені системи потребують інсталяції, але перед тим як завантажити застосунок, його потрібно викласти в маркет від Apple, так як сторонні додатки користувачі iOS та iPadOS не мають змоги.

Після запуску користувач отримає доступ до авторизації та реєстрації, а після цього зможе використовувати основні модулі додатку.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними для розробленого додатка є дані, які зчитуються з UITextField та дії користувача (наприклад підтвердження відповіді, як вірної)

Вихідними даними є оновлення користувацького інтерфейса з відповідними змінами після дій користувача.